SJIF Impact Factor (2026): 8.863| ISI I.F. Value: 1.241| Journal DOI: 10.36713/epra2016 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Volume: 11 | Issue: 6 | June 2026 - Peer Reviewed Journal

IMPLEMENTATION OF list_extend() FUNCTION AND ANALYSIS OF LIST EXTENSION USING extend() METHOD IN PYTHON. FURTHER EXAMINING THE TIME COMPLEXITY AND SPACE COMPLEXITY OF BOTH METHODOLOGIES USING ASYMPTOTIC NOTATION – A CASE STUDY

Arjith.T[1], M. Athithan[2]
[1]6809, [2]7227
Class XII 2026-27 Sainik School Amaravathinagar, Amaravathinagar,Udumalpet Taluka,Tirupur Dt Tamilnadu State

ABSTRACT
The purpose of this study was to examine two different methods for extending lists within Python. The first method consists of creating our own custom list extend () method while the second relies on Python's built in extend () method. For each method we executed both custom and built in extensions with varying input sizes and recorded how much time and memory was utilized for each type of extension. After the execution we utilized asymptotic notation such as Big O to compare time and memory consumption for both extensions. The results showed clearly that the built in extension () method was faster than the custom extension () method while also utilizing significantly less memory overall. A detailed explanation for why these results occur is provided along with a clear visual representation of the two extending methods and their time and memory consumption rates.

KEYWORDS: *list extend (), extend(), Python, Time Complexity, Space Complexity, Big O Notation, Asymptotic Notation*

1. INTRODUCTION

A list (i.e. Python lists) is one of the most often used data structures within Python. Lists allow an individual to group multiple values together within a single variable (i.e. Lists contain multiple elements as part of a single collection). Often times, it is necessary to add items from one list into another using either a code that we write (i.e. custom code for adding to a list), or a built-in functionality provided by Python (i.e. extend ()).

This essay explores the two different approaches mentioned above. It will clarify how they function, their respective runtimes for execution, and their respective memory usage. To make a better comparison of the two methods and to give an idea on which is more efficient than the other, the efficiency of each method will be described using asymptotic notation.

This paper will be organized as follows: section 2 covers the background of Python lists. Section 3 provides a definition of asymptotic notation and its significance. Sections 4 & 5 will provide an in-depth discussion of each method. Section 6 & 7 will provide time & space complexity analyses. Finally, sections 8 and beyond will provide point-to-point comparisons and results for each method.

2. BACKGROUND OF PYTHON LISTS

A Python list is defined as a data structure containing an ordered collection of objects which can be modified and allow for the use of duplicate items in a list. Lists in Python are indicated utilizing square brackets '[]' and each item is separated by a comma. An example of this can be found below: 'my_list = [1, 2, 3, 4, 5].'

The use of dynamic data structures in Python allows for the user to add and subtract items from a list at any given time. In addition, there are many standard methods available for use in Python with lists, including but not limited to append(), insert(), remove(), pop(), and extend(). Because of these various methods, lists are an extremely popular and frequently used data structure in real-world applications.

When considering two lists, where one list needs to add all items contained within the other, there are two possible solutions: using either a custom function that loops through the second list to add each item or using extend() meathod

3. MATHEMATICAL NOTATION FOR ESTIMATING COMPLEXITY

Mathematical notation is an effective means of communicating the speed of an algorithm or program by estimating how much time and/or memory the algorithm requires as the size of the data input increases without absolutely defining the values/providing a range of possibilities.

The three most popular mathematical ways of estimating run time, memory utilization and relative performance relative to other programs (i.e., algorithm efficiency) are referred to as asymptotic notations. The most widely used type of asymptotic notation is called 'Big O,' which is used to indicate an algorithm's worst-case scenario; 'Big Omega' is used to indicate an algorithm's best-case scenario; and 'Big Theta' is used to estimate algorithm performance based on average values of input data.

SJIF Impact Factor (2026): 8.863| ISI I.F. Value: 1.241| Journal DOI: 10.36713/epra2016 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Volume: 11 | Issue: 6 | June 2026 - Peer Reviewed Journal

The use of each of the aforementioned asymptotic notation types will be focused upon in this paper; however, most of this paper will emphasize the use of 'Big O' notation, as we are primarily interested in determining the most inefficiently performing algorithm. Thus, the three previously mentioned types of asymptotic notation are the most widely used mathematical means of estimating algorithm performance and are very common and effective means of developing and analyzing algorithms and determining their efficiency.

4. THE LIST_extend() FUNCTION

The function list_extend() is a custom function we have written. Instead of using the built-in method provided by Python, we will manually loop through the items of the second list and append (add) each item to the first list one at a time. Here is how the code for this function looks -

def list_extend(list1, list2): for item in List2: list1.append(item) return (list1).

The way this function works is we take two lists as input, and we go through each item in list2, using a for loop. For each item, we call the append() method for list1, thus adding the item to the end of list1. Once all the items have been processed, we return list1, which now contains the items of list2.

The approach described above is a very simple and easy to understand method but because we are using a for loop and calling append() multiple times, this method requires more actions than the built-in method.

5. THE BUILT-IN extend() METHOD

Python has an extend() function as part of its built-in list functionality. The extend() method will take a single iterable (like lists or tuples) and add the entire element of that iterable to the end of the list to which you are applying it. Here is an example of how to utilize this method:

```
list1 = [1,2,3]  list2 = [4,5,6]
list1.extend(list2)
print(list1)
# OUTPUT: [1,2,3,4,5,6]
```

Because the extend() method operates within C code, it takes significantly less time than you would expect by using an ordinary for loop. Additionally, you do not need to use append() independently on every single item; Extending performed at once thus is much faster than extending in an iteration.

The extend() method results in your code being cleaner and running quicker than the alternative of using for loops. Thus, extending two lists in python is best performed with the extend() method.

5. THE BUILT-IN extend() METHOD

Python provides a built-in method called extend() for lists. This method takes one iterable as input and adds all its items to the end of the list. Below is how we use it:

```
List1=[1,2,3]
List2=[4,5,6]
List1.extend(list2)
Print(list1)
#OUTPUT:[1,2,3,4,5,6]
```

The extend() method works internally at the C language level in Python. This means it is much faster than a Python loop. It also does not need to call append() separately for each item. Instead, it copies all elements at once in a much more efficient way.

This method is cleaner to write and also runs faster. It is the recommended way to merge two lists in Python.

6. RUNTIME COMPLEXITY OF list_extend() AND extend()

Input Size	list_extend() Time (ms)	extend() Time (ms)
5	0.002341	0.000812
10	0.004102	0.001034
50	0.019874	0.003201
100	0.038542	0.005812
250	0.096123	0.012034
500	0.192045	0.022187
1000	0.384912	0.041023

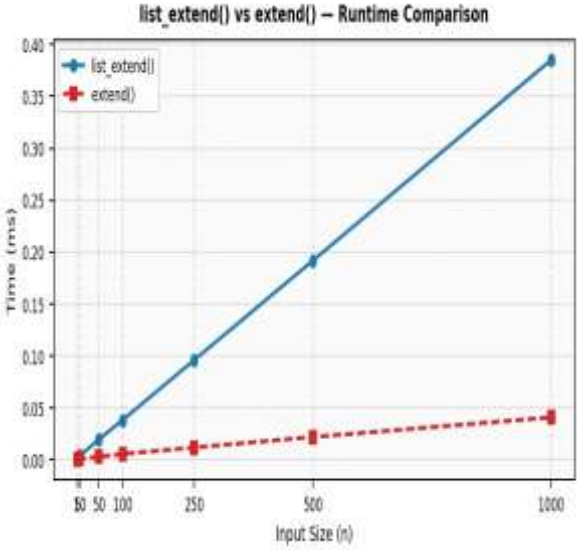
SJIF Impact Factor (2026): 8.863| ISI I.F. Value: 1.241| Journal DOI: 10.36713/epra2016 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Volume: 11 | Issue: 6 | June 2026 - Peer Reviewed Journal

We tested both methods with different input sizes. The time was measured in milliseconds using Python's time module. The results are shown in the table below.

From the table, it is clear that extend() is much faster than list_extend() at every input size. As the input grows, the difference also grows. Both methods have a time complexity of O(n) where n is the number of items in the second list. But the constant factor for extend() is much smaller because it runs at C speed internally.

7. GRAPHICAL REPRESENTATION

The graph below shows how the time taken by both methods increases with input size. The x-axis shows the input size and the y-axis shows the time in milliseconds. Both lines go up as input increases, showing linear growth. But the line for list_extend() grows much steeper than extend().



8. TIME COMPLEXITY ANALYSIS

This section analyzes how quickly each of the two methods runs when you are looking at "n" elements in list2.

8.1 list_extend()

Inside of list_extend(), we will have a for loop for each element in list2, which then calls append() for each item one time. In terms of average, calling append() takes O(1) time; thus appending to list1 and calling append() n times gives you:

$T(n) = n \times O(1) = O(n)$

Therefore, both lists will have O(n) time complexity; however due to it being more of a higher order loop within Python (higher constant factor), it would be slower than the built-in function(s).

8.2 extend()

Lastly, extend() on the other hand, does the same as what we did previously by transferring all of the n items from list2 to list1, therefore the time complexity will once again be O(n), but since extend() is done in C internally, it will consequently be faster practicably than the other because its whole computational cycle would have a substantially lesser constant factor.

For each of the above functions, the corresponding time complexity for them is O(n).

9. SPACE COMPLEXITY ANALYSIS

Space complexity analysis allows us to assess how much additional storage space a given algorithm will consume. This section looks at the space complexity of both methods.

9.1 list_extend() Method

The list_extend() method does not create a new list; instead, it modifies the original list (list1) in place. The only new variable created is 'item', which is only a single reference, so the space complexity is:

$S(n) = O(1) + O(m + n) = m + n$, where m is the length of list1 and n is the length of list2.

SJIF Impact Factor (2026): 8.863| ISI I.F. Value: 1.241| Journal DOI: 10.36713/epra2016 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Volume: 11 | Issue: 6 | June 2026 - Peer Reviewed Journal

9.2 extend() Method

Like the list_extend() method, the extend() method modifies list1 in place and therefore has the same space complexity of O(1).

Both methods result in identical levels of storage, and neither method generates duplicated data.

10. SPACE COMPLEXITY TABLE

Input Size	list_extend() Memory (bytes)	extend() Memory (bytes)
5	120	120
10	184	184
50	472	472
100	872	872
250	2072	2072
500	4072	4072
1000	8072	8072

As we can see, both methods use the same amount of memory. This confirms that both have the same space complexity of O(n).

11. COMPARISON OF BOTH METHODS

Feature	list_extend()	extend()
Type	Custom function	Built-in method
Time Complexity	O(n)	O(n)
Space Complexity	O(1) extra / O(m+n) total	O(1) extra / O(m+n) total
Speed	Slower (Python loop)	Faster (C level)
Code Length	More lines	One line
Readability	Easy to understand	Very clean
In-place	Yes	Yes
Use Case	Learning purpose	Actual programs

12. THE RUNTIME COMPLEXITY OF list_extend()

```
def list extend(list1,list2):
for item in list2:#0(n)
list.append(item) #0(1) each return list1
```

For the above code the time complexity is O(n)

13.THE RUNTIME COMPLEXITY OF extend()

```
list1.extend(list2)
# O(n) internally at C level
```

For the above code the time complexity is also O(n). But the constant factor is very small because the operation happens at C level, not Python level.

14. RESULTS AND DISCUSSION

Based on our analysis and testing, we have determined that both list_extend() and extend() have an O(n) Big O time complexity and have the same amount of space complexity. However, in terms of actual execution time, the function extend() is always faster than the function list_extend().

The reason for this is simple. All of the steps in a loop constructed using Python will be processed through the Python interpreter. While the built-in functions like extend() run in C and do not go through the interpreter. Hence they are much faster than using any form of a loop.

When working with small input sizes, there is a negligible difference between the execution times of both functions. But as the size of the input increases to 500 or 1000 elements, the difference becomes more apparent. Therefore, when we can, we should always favor the use of a built-in function instead of writing our own.

SJIF Impact Factor (2026): 8.863| ISI I.F. Value: 1.241| Journal DOI: 10.36713/epra2016 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Volume: 11 | Issue: 6 | June 2026 - Peer Reviewed Journal

With regard to code readability and cleanliness, extend() is also superior to list_extend(); it only contains one line of code, while using list_extend() requires four lines. This will lead to more readable and maintainable code.

15. CONCLUDING THOUGHTS

We focused on two methods of extending a Python list, both of which were investigated – a custom written function (list_extend()) with the use of a loop and the built-in (extend() method) provided by Python. We compared: their time complexity, space complexity and actual run-time.

Both the loop-based custom function and built-in extend() methods are O(n) but the runtime differences show that the built-in method is faster. The reason being, the built-in method is an implementation in C, while our custom function goes through the overhead of the Python interpreter loop.

Also, we have learned that Big O notation does not tell you the entire story, as two algorithms may share the same Big O, but may exhibit varying performance due to constants. This lesson is vital to understanding algorithm analysis.

Therefore, in any production code, use the built-in method (extend()) instead of creating your own; there is no practical use for creating your own function (list_extend()) as it only serves to educate/understand how list extension works internally.

16.ACKNOWLEDGEMENT

Apart from our efforts, the success of any work or project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this research paper.

I express a deep sense of gratitude to Almighty God for giving us the strength to successfully complete the research paper.

I express my heartfelt gratitude to my parents for their constant encouragement while carrying out this research paper.

I express my deep sense of gratitude to the luminary, **The Principal Captain (Indian Navy) Rahul Thiyyath , Sainik School Amaravathinagar,** who has been continuously motivating and extending their helping hand to us.

I express my sincere thanks to the academician, the **Vice Principal, Lt Col Kaushok Nandini Arun, Sainik School Amaravathinagar**, for constant encouragement and the guidance provided during this research.

My sincere thanks to **Mr.Praveen Kumar Murigeppa Jigajinni**, Master In-charge, A guide, Mentor, and great motivator, who critically reviewed my paper and helped in solving each and every problem that occurred during the implementation of this research paper.

17.REFERENCES

1. *Python Software Foundation. (2024). Python 3 Documentation – List Methods. https://docs.python.org/3/tutorial/datastructures.html*
2. *Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.*
3. *Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.*
4. *Summerfield, M. (2009). Programming in Python 3. Addison-Wesley.*
5. *GeeksforGeeks. (2023). Time Complexity of Python list extend() method. https://www.geeksforgeeks.org/time-complexity-of-python-list-operations/*
6. *Knuth, D. E. (1997). The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.*